National College of Ireland

Cryptocurrency Price Prediction Using Temporal Decay Attention LSTM: A Novel Approach for Time-Series Forecasting

MSc Research Project
Master Of Science in Data Analytics

Krishnaseshu Kavadi
X23329050

School of Computing
National College of Ireland

Supervisor: Abubakr Siddig

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Krishnaseshu Kavadi
Student ID:	X23329050
Programme:	MSCDAD_A **Year:** 2024-2025
Module:	Research Practicum
Lecturer:	Abubakr Siddig
Submission Due Date:	11 Aug 2025
Project Title:	Cryptocurrency Price Prediction Using Temporal Decay Attention LSTM: A Novel Approach for Time-Series Forecasting
Word Count:	844 **Page Count:** 9

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Krishnaseshu Kavadi

Date: 10 Aug 2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	

Penalty Applied (if applicable):	

Cryptocurrency Price Prediction Using Temporal Decay Attention LSTM: A Novel Approach for Time-Series Forecasting

Krishnaseshu Kavadi
X23329050

1. Introduction

This configuration manual provides a detailed technical roadmap for setting up the environment, dependencies, and implementation procedures required for this research project.

2. System Hardware Requirements

To successfully implement and run the Temporal Decay Attention LSTM model for cryptocurrency price forecasting, the following minimum system specifications are recommended:

- **Operating System:**
  - *Preferred:* Windows 10 or 11 – fully compatible with the required Python libraries and deep learning frameworks.
  - *Alternative:* Ubuntu 20.04+ – suitable for users familiar with Linux-based environments.
- **Processor (CPU):**
  - Intel Core i5 / AMD Ryzen 5 or higher.
  - A quad-core processor or better is recommended to ensure efficient data processing and model training.
- **Memory (RAM):**
  - *Minimum:* 16 GB
  - *Recommended:* 32 GB – especially useful when working with large historical price datasets and during parallel computation tasks.
- **Graphics Processing Unit (GPU):**
  - An NVIDIA GPU with CUDA support is strongly advised for accelerating model training using deep learning frameworks like TensorFlow or PyTorch.

- **Storage:**
    - At least 15–20 GB of free disk space to accommodate raw datasets, environment dependencies, trained model checkpoints, and result files.

3. Software Requirements:

To build and execute the Temporal Decay Attention LSTM model for cryptocurrency price prediction, a suitable software environment is essential

Model development relies on deep learning libraries, primarily:

- **PyTorch (`torch`)** – used for building and training the LSTM-based architecture.
- **Torch Geometric (`torch_geometric`)** – optionally included for future extensions to graph-based models.
- **Torchvision** – used occasionally for model component utilities, though not central to this study.
- **NetworkX** – helpful in exploring potential graph-based relationships if integrated later.

To streamline the training pipeline, utilities such as `DataLoader` and `TensorDataset` are used for batching and data feeding. Input features are normalized using `StandardScaler`, and model optimization is carried out using `CrossEntropyLoss`, `Adam`, or `AdamW` from PyTorch's optimization modules.

Development can be done in:

- **Jupyter Notebook** – ideal for interactive, cell-based experimentation.
- **Visual Studio Code (VS Code)** – a lightweight, highly customizable code editor.
- **PyCharm** – a full-featured IDE suitable for managing more complex workflows.

For managing the environment, it is recommended to use **conda** to create isolated, reproducible setups, which is especially beneficial when dealing with deep learning dependencies and GPU drivers.

4. Dataset Details

The dataset employed in this research comprises hourly cryptocurrency market data, serving as the foundation for training and evaluating the Temporal Decay Attention LSTM model.

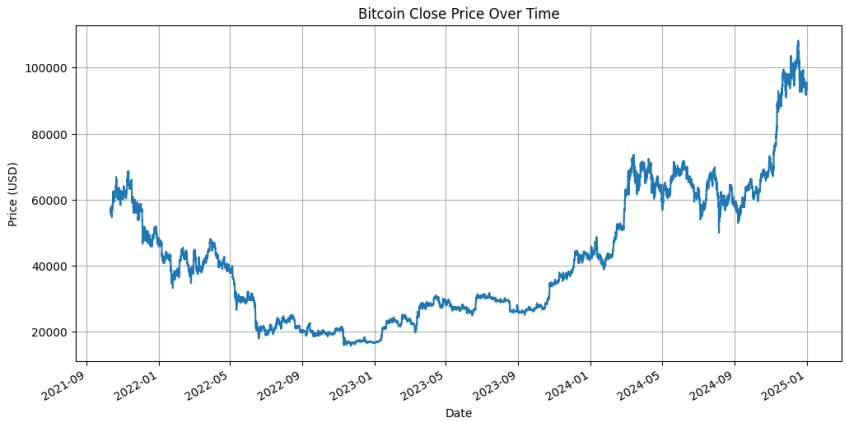

Figure 1. Bitcoin Close Price Over Time

Preprocessing Steps:

- Loaded **& cleaned** hourly Bitcoin data (2021–2024); set datetime index.
- Dropped **irrelevant columns**; confirmed key trading fields are complete.
- Visualized **sentiment vs. price** to explore market behavior.
- Engineered **features** (lags, rolling stats, sentiment smoothing).
- Normalized **data** and created 240-hour sequences for LSTM forecasting.

5. Model Configuration

a) AdaBoost Regressor

```
from sklearn.ensemble import AdaBoostRegressor

# Train AdaBoost Regressor
ab_model = AdaBoostRegressor(random_state=42, n_estimators=100)
ab_model.fit(X_train_2d, y_train)

# Predict
ab_preds = ab_model.predict(X_test_2d)

# Evaluate
evaluate_model("AdaBoost Regressor", y_test, ab_preds)
```

b) Decision Tree Regressor

```python
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
import numpy as np



# Train Decision Tree
dt_model = DecisionTreeRegressor(random_state=42)
dt_model.fit(X_train_2d, y_train)
dt_preds = dt_model.predict(X_test_2d)

# Print metrics
evaluate_model("Decision Tree", y_test, dt_preds)
```

c) LightGBM Regressor

```python
import lightgbm as lgb
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Train LightGBM model
lgb_model = lgb.LGBMRegressor(random_state=42)
lgb_model.fit(X_train_2d, y_train)

# Predict
lgb_preds = lgb_model.predict(X_test_2d)

# Optional: define evaluation function if not already defined
def evaluate_model(name, y_true, y_pred):
    mae = mean_absolute_error(y_true, y_pred)
    mse = mean_squared_error(y_true, y_pred)
    rmse = np.sqrt(mse)
    r2 = r2_score(y_true, y_pred)
    print(f"\n{name} Evaluation:")
    print(f"MAE:  {mae:.6f}")
    print(f"MSE:  {mse:.6f}")
    print(f"RMSE: {rmse:.6f}")
    print(f"R²:   {r2:.4f}")

# Evaluate
evaluate_model("LightGBM", y_test, lgb_preds)
```

d) LSTM Model

```python
#  LSTM Model for Bitcoin Price Forecasting
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import numpy as np
import matplotlib.pyplot as plt

#  Define MAPE
def mean_absolute_percentage_error(y_true, y_pred):
    epsilon = np.finfo(np.float64).eps
    return np.mean(np.abs((y_true - y_pred.ravel()) / np.maximum(np.abs(y_true), epsilon))) * 100

#  Build LSTM model
model = Sequential()
model.add(LSTM(units=64, return_sequences=True, input_shape=(X_train.shape[1], X_train.shape[2])))
model.add(Dropout(0.7))
model.add(LSTM(units=32))
model.add(Dropout(0.7))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mean_squared_error', metrics=['mae'])
model.summary()
```

e) Temporal Decay Attention LSTM

```python
import tensorflow as tf
from tensorflow.keras.layers import Input, LSTM, Dense, Dropout, Layer
from tensorflow.keras.models import Model
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Define standard attention layer
class StandardAttention(Layer):
    def __init__(self, **kwargs):
        super(StandardAttention, self).__init__(**kwargs)

    def build(self, input_shape):
        self.W = self.add_weight(name='att_weight',
                                 shape=(input_shape[-1], 1),
                                 initializer='random_normal',
                                 trainable=True)
        self.b = self.add_weight(name='att_bias',
                                 shape=(input_shape[1], 1),
                                 initializer='zeros',
                                 trainable=True)
        super(StandardAttention, self).build(input_shape)

    def call(self, inputs):
        e = tf.matmul(inputs, self.W) + self.b           # (batch, time_steps, 1)
        e = tf.squeeze(e, axis=-1)                       # (batch, time_steps)
        alpha = tf.nn.softmax(e, axis=1)                 # (batch, time_steps)
        alpha_expanded = tf.expand_dims(alpha, axis=-1)  # (batch, time_steps, 1)
        context = tf.reduce_sum(inputs * alpha_expanded, axis=1)  # (batch, features)
        return context

# Build model
def build_attention_lstm(input_shape):
    inp = Input(shape=input_shape)
    x = LSTM(64, return_sequences=True)(inp)
    x = Dropout(0.75)(x)
    x = LSTM(32, return_sequences=True)(x)
```

6. Evaluation and Results

6.1 Evaluation Metrics:

To evaluate model performance effectively, the following regression metrics were used:

R^2 Score,MAE,MSE,RMSE,MAPE

6.2 Final Results:

```
 Temporal Decay Attention LSTM Evaluation:
MAE:  0.020244
MSE:  0.000700
RMSE: 0.026456
MAPE: 3.45%
R²:   0.9702
```

References

Scikit-learn (Machine Learning Library):https://scikit-learn.org/stable/documentation.html

Matplotlib (Python Visualization Library):https://matplotlib.org/stable/contents.html

Seaborn (Statistical Data Visualization):https://seaborn.pydata.org/

Statsmodels (Statistical Modeling Library):https://www.statsmodels.org/stable/index.html

PyCaret (Low-code ML Library):https://pycaret.gitbook.io/docs/

Keras Attention Layer Guide (for Custom Attention Mechanisms):
https://keras.io/examples/nlp/transformer/

Google Colab (Cloud-based Python Notebook Environment):https://research.google.com/colaboratory/

Pytorch (Deep Learning Framework):https://pytorch.org/docs/stable/index.html